CONOSCERE
IL COMPUTER DIRETTAMENTE DAL COMPUTER

per Commodore Vic20 e 64

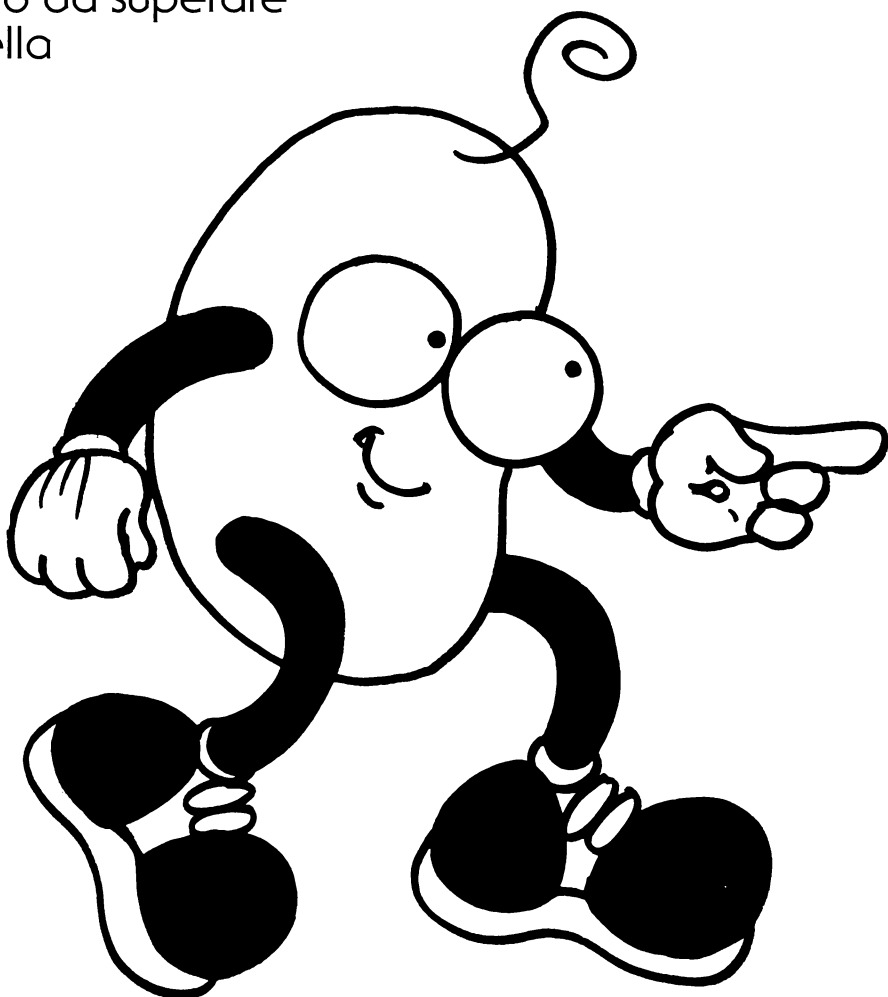
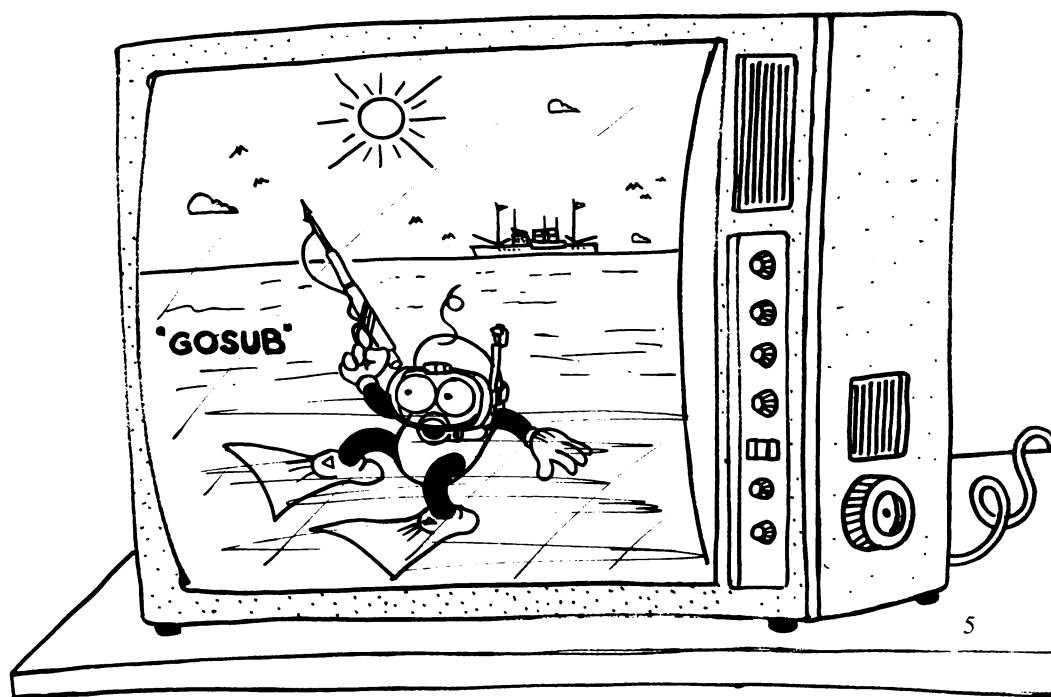
Beatrice d'Este

In questa lezione vediamo come individuare gli errori più frequenti che si possono verificare nei listati, cercando di capirne la causa e quindi il modo migliore per correggerli.
Se il computer, durante l'esecuzione di un programma, trova un errore nel listato ferma subito l'esecuzione e visualizza un messaggio seguito dal numero di linea in cui è stato trovato l'errore.
In questo caso, prima di tutto, devi listare la linea in cui ti viene segnalato l'errore, scrivendo LIST seguito dal numero di linea.
Potrai quindi, posizionandoti con il cursore, correggere l'errore.
Se hai scritto una linea con molte istruzioni e non riesci a trovare l'errore, ti consiglio di dividerla in più linee, poi con il RUN rimanda in esecuzione il programma e guarda in quale delle nuove linee verrà segnalato il messaggio di errore. In questo modo, visto che dovrai controllare un numero minore di istruzioni, ti sarà più facile individuarlo.

ELENCO ERRORI

SYNTAX ERROR
TYPE MISMATCH
NEXT WITHOUT FOR
RETURN WITHOUT GOSUB
UNDEF'D STATEMENT
BAD SUBSCRIPT
REDIM'D ARRAY
STRING TOO LONG
OUT OF DATA
ILLEGAL QUANTITY
OUT OF MEMORY
REDO FROM START
EXTRA IGNORED
DIVISION BY ZERO
FORMULA TOO COMPLEX
OVERFLOW
UNDEF'D FUNCTION
ILLEGAL DIRECT
BAD DATA
FILE NOT OPEN
FILE OPEN
NOT OUTPUT FILE
NOT INPUT FILE
VERIFY
LOAD

Tieni presente che, dopo il verificarsi di un errore, o dopo aver fermato l'esecuzione del programma con il RUN/STOP, potrai conoscere il contenuto di una certa variabile scrivendo in modo diretto PRINT, seguito dal nome della variabile e battere RETURN. Questo ti sarà utile per capire se l'errore è causato dal contenuto di qualche variabile che supera i limiti previsti.
Il messaggio di errore più frequente è **SYNTAX ERROR**, dovuto alla presenza di qualche istruzione che il computer non riesce a capire. Questo è causato principalmente da un errore di battitura (ad esempio: PRUNT invece di PRINT), dalla mancanza dei due punti per separare due istruzioni, dalla mancanza delle parentesi oppure dalla punteggiatura sbagliata.

L'errore **NEXT WITHOUT FOR** invece si verifica quando viene trovata la chiusura (NEXT) di un ciclo che non è stato aperto. In questo caso bisogna controllare che esista l'apertura (FOR) del ciclo prima del NEXT e se la variabile corrente presente nel FOR è la stessa del NEXT. Se stai usando più cicli annidati fai attenzione che siano effettivamente l'uno all'interno dell'altro.
Usando i vettori o le matrici ti potrà capitare l'errore **BAD SUBSCRIPT**, dovuto al fatto che il valore dell'indice di un elemento è superiore a quello massimo dichiarato nell'istruzione DIM. Quindi controlla che il vettore o la matrice siano stati dimensionati. In caso contrario ricorda che la dimensione standard, data dal computer, è di 10 elementi. Altrimenti, se l'indice è una variabile, dovrai controllare la logica del programma e vedere dove il valore dell'indice incrementa, in modo da superare il limite massimo della dichiarazione.

Se dimensioni due volte lo stesso vettore, ti verrà mostrato **REDIM'D ARRAY** e in tal caso dovrai eliminare una delle due istruzioni DIM.
Con l'uso delle subroutine ti potrà capitare invece l'errore **RETURN WITHOUT GOSUB**, dovuto al fatto che il programma in esecuzione ha incontrato l'istruzione RETURN senza avere prima compiuto un salto alla subroutine con il GOSUB.
In questo caso bisognerà controllare che nel programma non ci siano salti alla subroutine eseguiti dal GOTO invece che dal GOSUB, assicurarsi che il programma principale abbia un punto di fine e che l'esecuzione non sconfini nella prima subroutine accodata al programma principale.
Durante la concatenazione di più stringhe, se cerchi di generare una stringa con oltre 255 caratteri, può verificarsi uno **STRING TOO LONG**. In tal caso ti consiglio di aggiungere nel programma le istruzioni LEN per controllare le lunghezze delle stringhe prima che vengano concatenate.
Quando viene usato un dato numerico al posto di una stringa o viceversa (ad esempio: A$ = 45, R = "F", E$ = VAL(Y$)) il computer segnala **TYPE MISMATCH**. In questo caso controlla bene le assegnazioni, le condizioni, le funzioni stringa e quelle matematiche.

L'errore **UNDEF'D STATEMENT** ti verrà mostrato se cerchi di compiere un salto ad un numero di linea inesistente. Se stai usando le istruzioni READ e DATA e si verifica un **OUT OF DATA**, vuol dire che è stato eseguito un READ ma che i dati da leggere sono finiti.
In questo caso controlla che tutti i dati siano correttamente separati con la virgola e che il numero dei READ eseguiti sia uguale al numero dei DATA.
L'**OUT OF MEMORY** viene visualizzato quando non c'è più memoria sufficiente a contenere il programma o le variabili, oppure quando troppi cicli o troppe subroutine sono annidate.
L'**EXTRA IGNORED**, a differenza degli altri errori, non è dovuto ad uno sbaglio nel listato, ma si verifica se alla richiesta di un INPUT si scrive la risposta con all'interno una virgola o un due punti. In tal caso l'INPUT accetterà solo i caratteri che precedono la punteggiatura e quindi il programma proseguirà l'esecuzione.
Il **REDO FROM START** si verifica se in un INPUT, alla richiesta di un dato numerico, vengono invece inseriti dei caratteri. Basterà comunque ribattere l'inserimento corretto e il programma continuerà l'esecuzione.

Listato dell'esercizio: ESEMPIO PRATICO

```
10 print"quiz moltiplicazioni"
20 goto200
30 input"difficolta' (1-10)";a
40 if(a<1)+(a>10)then15
50 a=int(a/2+0.5)
60 gosub200
70 imput"quante domande";b
75 ifb<1then70
80 forg=1tob:print""
85 c=a*int(10*rnd(0)+1)
90 d=a*int(10*rnd(0)+1)
95 e=c*d
100 print"domanda n.";g
110 gsub200:print"cosa fa"c"x"d
120 input"";f
125 iff=etenprint"   esatto":h=h+1:goto140
130 print"   sbagliato":print"risposta esatta:";e
140 print"   punteggio";h
150 prunt"premi il return"
160 gety$:ify$<>chr$(13)then165
170 nextw
180 print"   fine quiz"
185 print"risposte esatte"
190 print"   "h"su"b
200 print"":return
```

Listato dell'esercizio: COMPLETA IL LISTATO

```
10 printchr$(147):a$([*])="N":a$(2)="M"
20 for[*]to7:forK=1to5
30 t(1)=[*]
40 c$=a$(int(rnd(0)*[*]+1))
50 d$=[*](int(rnd(0)*2+1))
60 print" "c$" "d$;
70 if(e$=a$(1))*(c$=a$(2))thens=[*]+1
80 e$=d$
90 [*](c$=a$(1))*(d$=a$(2))thens=s+1
100 nextK:print:[*]:nextx:print
110 input"sequenze N M";[*]
120 t(2)=ti
130 ifp<>sthenprint"sbagliato":[*]
140 print"esatto"
150 i=int((t(2)-t(1))/[*])
160 print"secondi impiegati:";[*]
```

PROGRAMMIAMO INSIEME (CBM 64)

```
10 poke53280,3:poke53281,3
20 g$="                "
30 print"   ora attuale"
40 input" (oommss)";p$
50 l=len(p$):ifl<>6then30
60 print"            ":t=11
70 printtab(t)"┌─────────────────┐"
80 printtab(t)"│┌───────────────┐│"
90 printtab(t)"││               ││"
100 printtab(t)"││               ││"
110 printtab(t)"││               ││"
120 printtab(t)"│└───────────────┘│"
130 printtab(t)"│┌┬┬┬┬┬┐          │"
140 printtab(t)"│└┴┴┴┴┴┘          │"
150 printtab(t)"└─────────────────┘"
160 ti$=p$
170 p$=left$(ti$,2)+":"+mid$(ti$,3,2)+":"+right$(ti$,2)
180 printg$tab(t+5)" "p$
190 ford=1to100:nextd
200 printg$tab(t+5)" "p$" "
210 ford=1to100:nextd
220 goto170
```

PROGRAMMIAMO INSIEME (VIC 20)

Il listato è identico a quello per il CBM 64,
tranne nelle linee:

```
10 poke36879,59
60 print"            ":t=1
```

Soluzione dell'esercizio: COMPLETA IL LISTATO

(lez. 22)

```
10 dimd(10)
20 fork=1to14:a$=a$+chr$(157):nextk
30 printchr$(147)"batti gli importi"
40 fork=1to10
50 print" "d(k)left$(a$,len(str$(d(k)))+2);
60 inputd(k):printchr$(145);
70 printright$("           "+str$(d(k)),11)"
80 t=t+d(k)
90 nextk
100 print"---------------"
110 printright$("           "+str$(t),11)
120 print:print"corretto (s/n)?"
130 gety$
140 ify$="s"thenend
150 ify$="n"thent=0:goto30
160 goto130
```